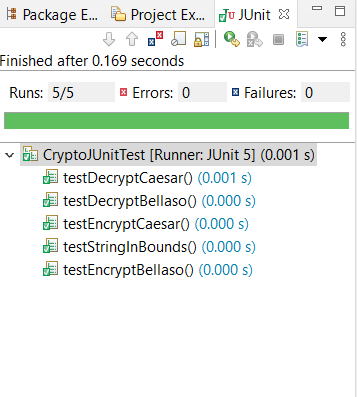
John Mobarry

CMSC 203

**Assignment 3 Learning Experience**

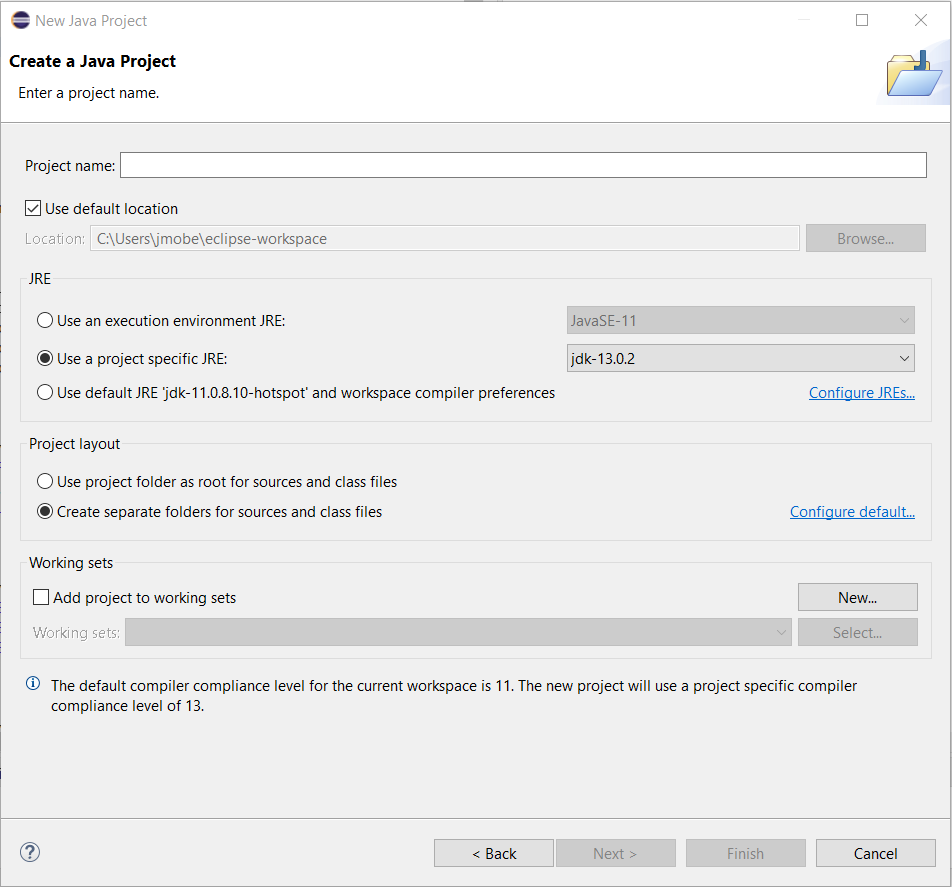
Assignment 3 was a learning experience that helped reinforce my understanding of topics ranging from JUnit Testing, correctly running JavaFX in eclipse, in combination with the implementation of methods we have learned in class. This assignment was a rigorous two week experience wrought with plenty of debugging, software troubleshooting, and retracing my steps through the course content to reach clarity on certain topics. In this learning experience reflection I will break down my experiences in JUnit testing, getting JavaFX to run correctly, and the methods in the assignment respectively.

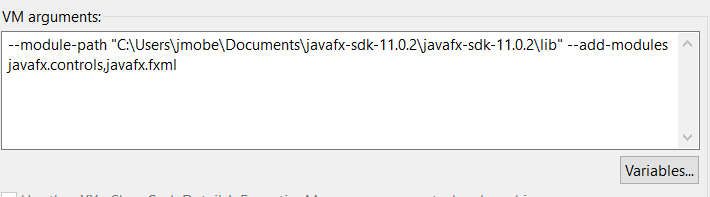
In previous assignments we have gone through the process of ensuring our code functioned efficiently by manually creating method outputs and checking if the outputs match what is expected. This can be a laborious process as it requires you to keep track in your program of what lines are for checks and need to subsequently be commented out or deleted for the final turn in of the project. The provided code “CryptoManagerTest.java” is a great example of the kind of testing that we would complete up to this point in order to confirm the validity of the code. Numerous method outputs are printed to the console to then be compared to the expected outputs, line by line. This is a perfectly acceptable way to confirm that the program is running correctly, but it is neither the most efficient or most elegant way. The introduction of JUnit Testing has created an excellent way of checking if the program is running as expected without manual confirmation. The JUnit Class is a class which hinges upon importing the JUnit package and creating setUP(), tearDown() and test() methods. The setUp methods are performed prior to every test and the tearDown methods are performed after every test method. The test methods are completed in random order, so they are not performed in the order that they appear in the .java file. In the test methods through the use of assertTrue, assertFalse, and assertEquals the eclipse IDE is able to confirm if outputs of methods match their expected outputs. When you run the JUnit code, it does not produce an output in the console like a class with main but it produces an output like:



The green check marks confirm that the outputs match the expected outputs and that your program is running correctly for the following methods. If the program were to not be running correctly, then there would be a blue x where the green check mark is. This method of debugging helps you identify issues directly through the code and save valuable programming time.

Getting JavaFX to run properly on the Eclipse IDE was definitely a learning experience as it was much more complicated than plug and play, since JavaFX is no longer natively supported in the more recent editions of Java. There were two major roadblocks when creating a JavaFX project that will run within the eclipse IDE: making sure that the project runs on the most recent version of Java instead of the Java native to the IDE and ensuring that the run configurations are set up correctly. You can easily hamstring yourself from getting the project to run correctly if you do not use the correct project specific JRE. For multiple days I was using jdk-11.0.08.10-hotspot instead of the desired jdk-13.0.02 which resulted in errors such as not being able to import the JavaFX application platform resulting in numerous errors.



Another issue that I struggled with after getting the Java Project set up in the correct JRE was that my run configurations were not working correctly for my project. I was mistakenly under the impression after completing the JavaFX lab that the configuration setup would then transfer over to any subsequent JavaFX projects I created. For multiple hours of programming I was frustrated in my inability to get the program running until I updated my VM arguments through Run Configurations → FxDriver → Arguments → VM arguments:

The methods that were created in the CryptoManager class were an excellent test of understanding regarding the use of for loops, while loops, and if statements in order to return boolean and String values from methods. The stringInBound() method is a required method to determine if the String entered into the method is either valid or not valid, to be returned via the boolean variable isValid. By using a for loop which increments through each character of the String, each character in the length of the String is checked to confirm if it is in bounds. If it is not isValid is returned as False. The encryptCaesar(), decryptCaesar(), encryptBellaso(), and decryptBellaso() all are methods that accept both a String and a key which are used within their Methods. To meet the required outputs I implemented while loops nested within for loops to scan the entire length of the string and complete the transformations, the while loops were to account for the ASCII ranges and to ensure that the encrypted and decrypted strings kept their values within the ranges. This was definitely the most time consuming and difficult of the projects that we have completed in our time in CMSC203, but it has definitely expanded my knowledge of the ECLIPSE IDE and using methods to complete complex problems I never previously thought I could attempt.